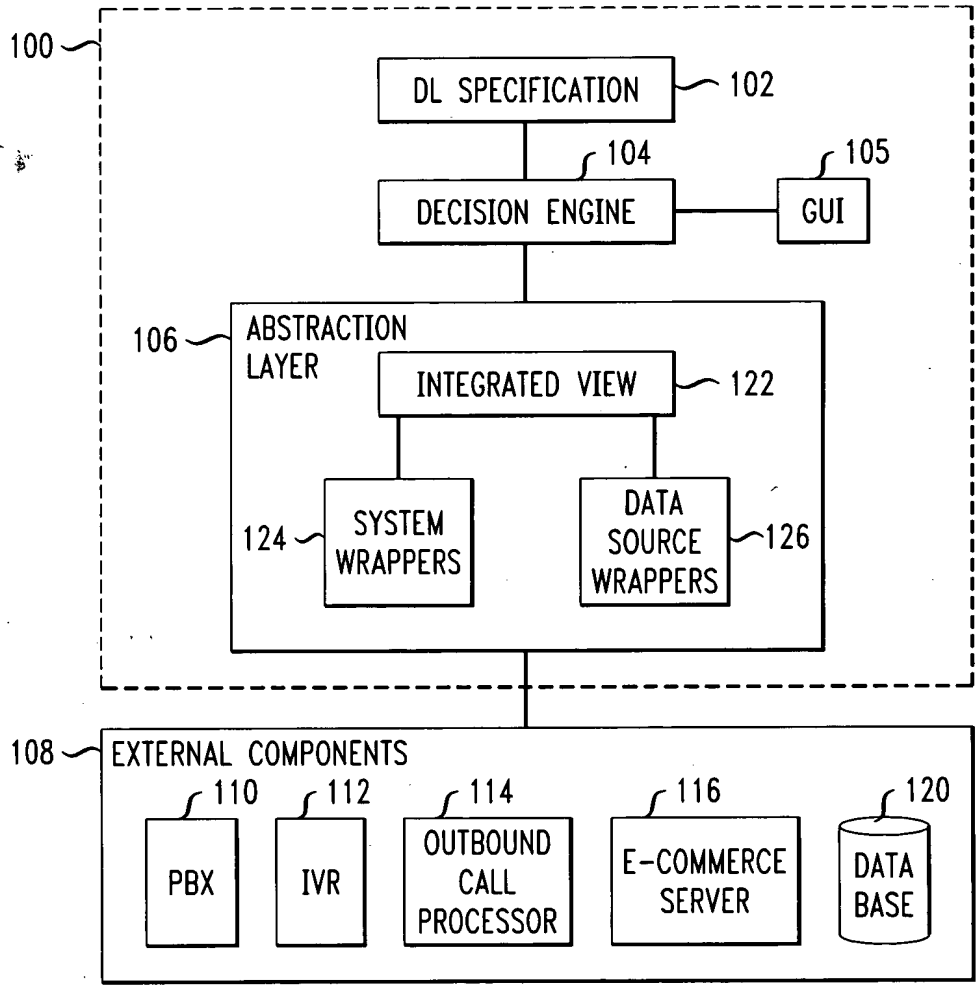
FIG. 1



09/251998

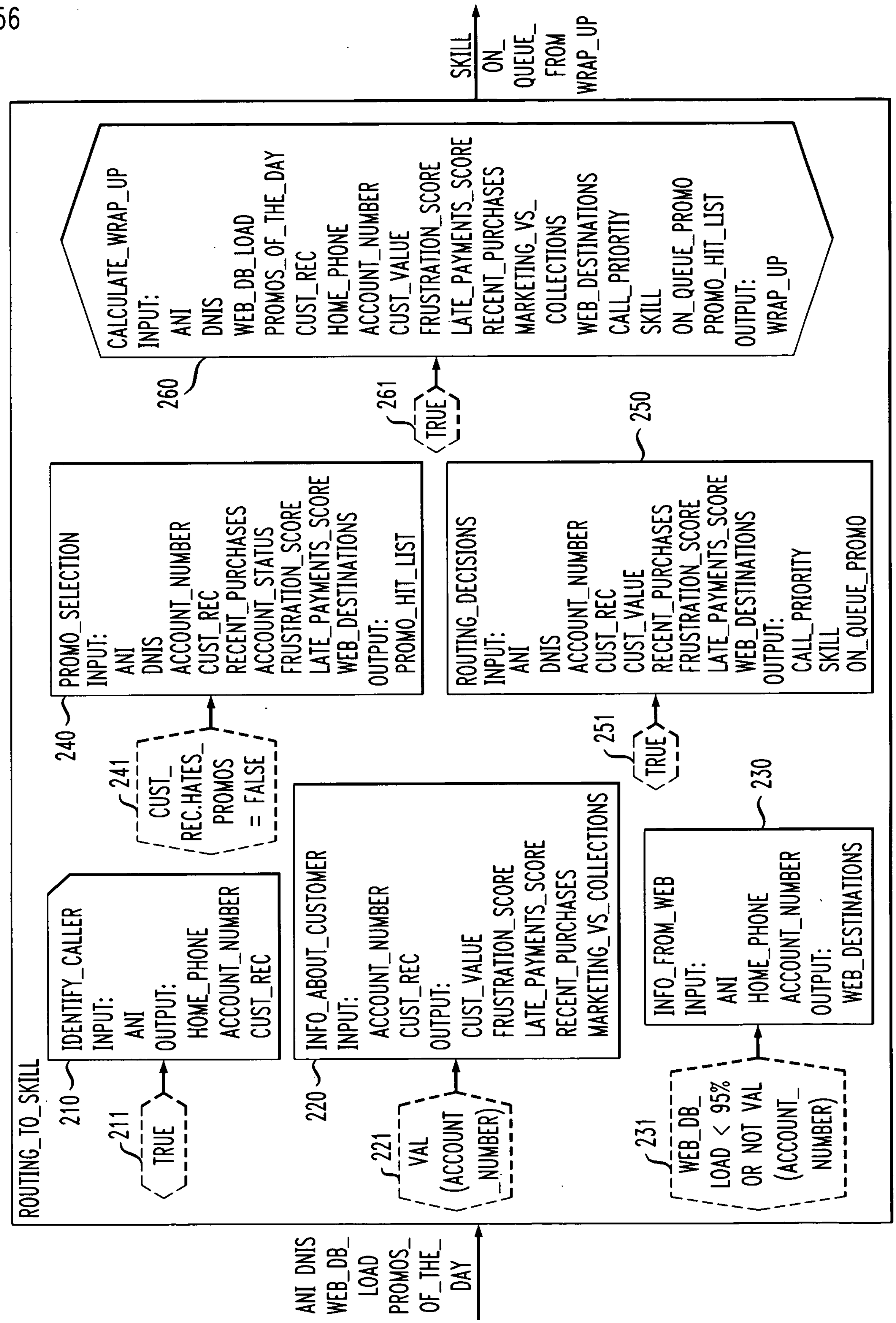


FIG. 2

FIG. 3

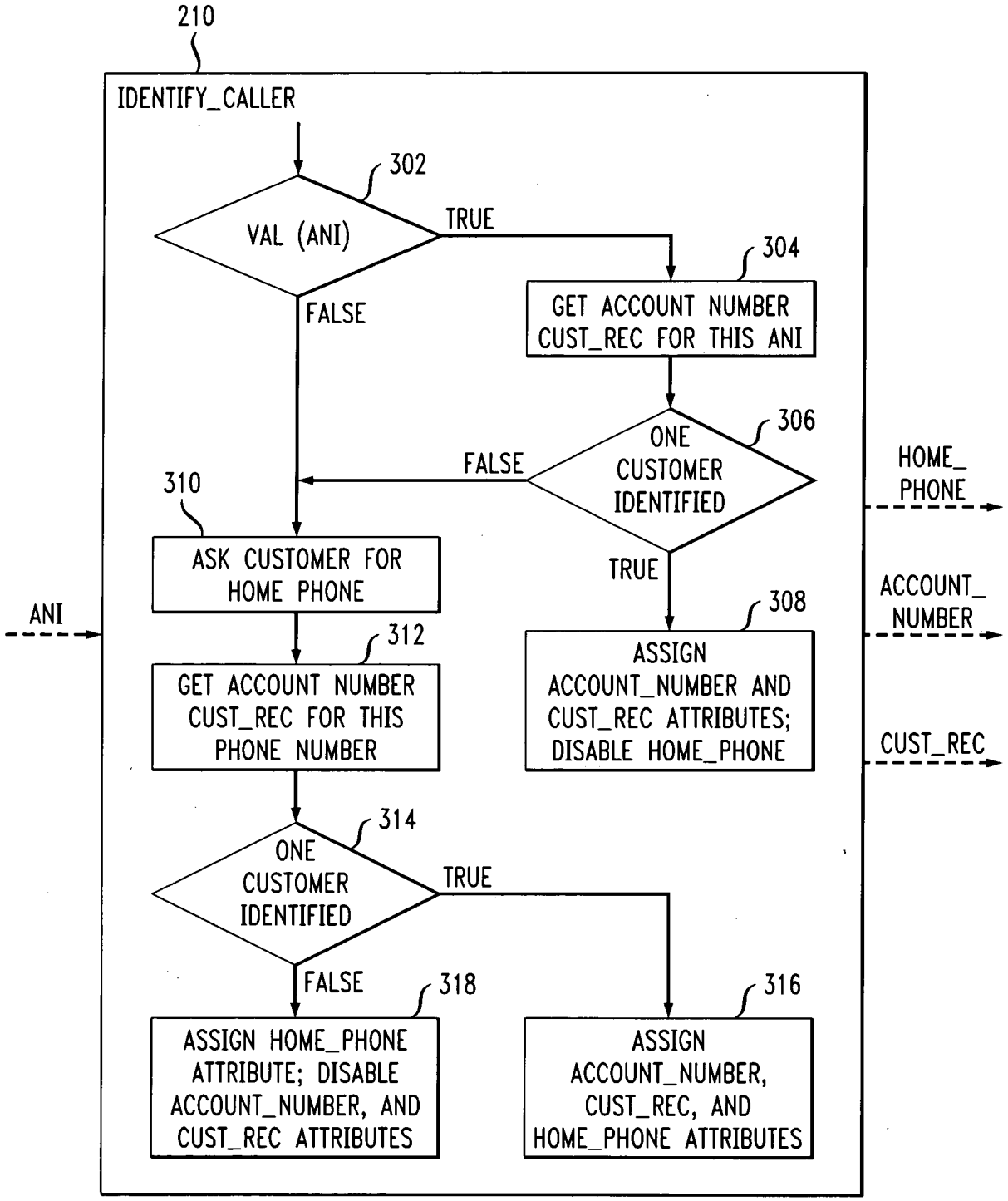


FIG. 4

```
1  Module:    identify_caller
2      Submodule of:    routing_to_skill
3      Input attributes:       ANI : 9digits
4      Output attributes:      home_phone : 9digits
5                              account_number : 15digits
6                              cust_rec : tuple (name: string,
7                                                address: string,
8                                                card_color: ("platinum",
9                                                "gold", "green"),
10                                               hates_promos? : boolean,
11                                               estimated_income_bracket :
12                                               ("0-10K", ">10K-20K",...,
13                                               ">100K-150K", ">150"),
14                                               last_sent_bonus_check:date)
15     Enabling condition:     true
16     Type:              flowchart
17     Computation:       See Fig. 3
18     Side-effect:       yes
19     Side Effect function:  (IVR Dip)
```

FIG. 5

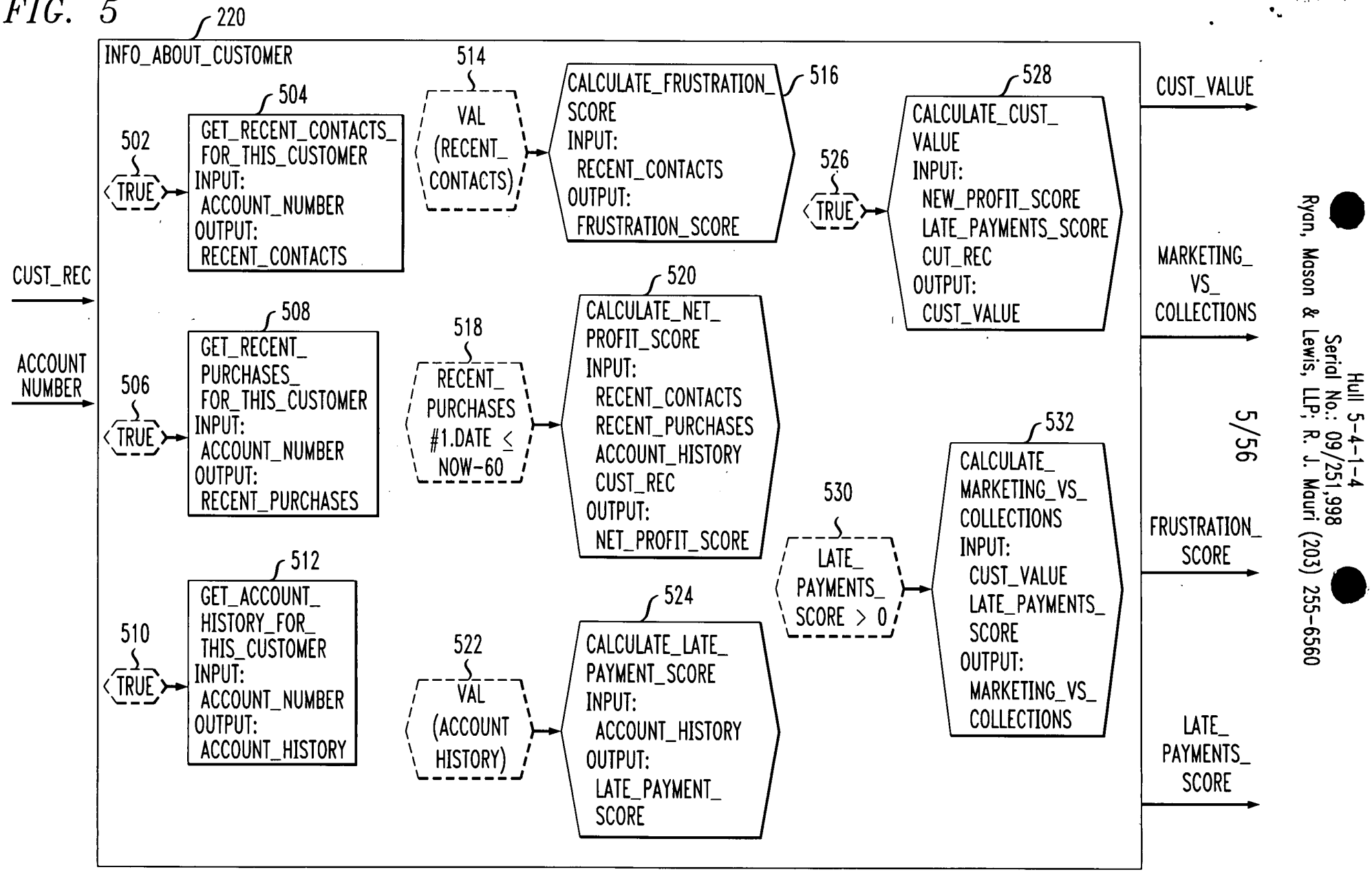


FIG. 6

```
 1 Module:    info_about_customer
 2     Submodule of:  routing_to_skill
 3     Input attributes:       account_number
 4                             cust_rec

 5     Output attributes:      cust_value : [1..10]
 6                             frustration_score : [1..10]
 7                             late_payments_score : [1..10]
 8                             recent_purchases :list(tuple( date : date,
 9                                                           item : 20digit,
10                                                           qty : int,
11                                                           amount: $value ))
12                             marketing_vs_collections : {"market",
13                             "collect"}
14
15 Enabling condition:         VAL(account_number)
16     Type:              declarative
17     Side-effect:       no
```

FIG. 7

```
 1 Module:    info_from_web
 2     Submodule of:  routing_to_skill
 3     Input attributes:       ANI
 4                             home_phone
 5                             account_number

 6     Output attributes:      web_destinations : list(tuple(regions: set of
 7                                                        {"Australia","Asia",...
 8                                                        "NAmerica-US", "US"},
 9                                                    itinerary:web_form_content,
10                                                    date_last_modified : date ))

11     Enabling condition:    web_db_load < 95% or not VAL(account_number)

12     Type:                  foreign

13     Computation:           get_web_data(ANI, home_phone, account_number)

14     Side-effect:           no
```

FIG. 8

```
1  Module:    promo_selection
2      Submodule of:   routing_to_skill
3      Input attributes:       ANI
4                              DNIS
5                              account_number
6                              cust_rec
7                              cust_value
8                              recent_purchases
9                              frustration_score
10                             late_payments_score
11                             web_destinations
12     Output attributes:      promo_hit_list : list ( promo_message )
13     Enabling condition:     cust_rec.hates_promos? = false
14     Type:                   foreign
15     Computation:            get_promo_hit_list(ANI, DNIS, account_number,
16                             cust_rec, cust_value, recent_purchases,
17                             account_status, frustration_score,
18                             late_payments_score, web_destinations)
19     Side-effect:            no
```

8551-52/50

FIG. 9

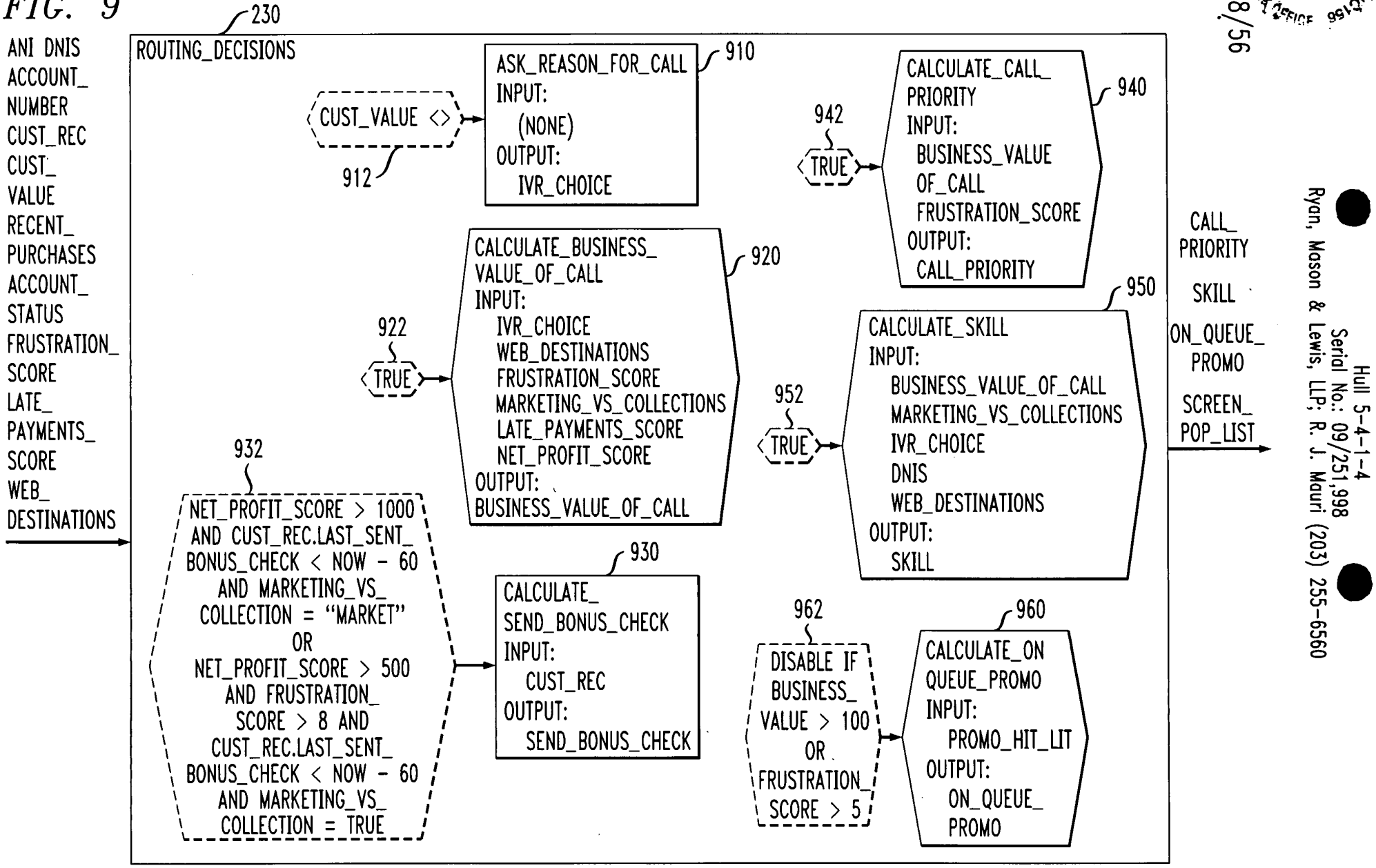


FIG. 10

```
1  Module:    routing_decisions
2      Submodule of:   routing_to_skill
3      Input attributes:       ANI
4                              DNIS
5                              account_number
6                              cust_rec
7                              cust_value
8                              recent_purchases
9                              frustration_score
10                             late_payments_score
11                             web_destinations
12     Output attributes:      call_priority : [1..4] \\corresponds to "low",
13                                    "med", "high", "top"
14                             skill : {"norm_tier_dom", "norm_tier_intl",
15                                    "australia_promo", "high_tier",
16                                         collections"}
17                             on_queue_promo : message_identifier
18                             screen_pop_list : list ( screen_entry )
19     Enabling condition:     true
20     Type:                   declarative
21     Side-effect:            yes
```

FIG. 11

```
1  Module:   calculate_wrap_up
2       Submodule of:    routing_to_skill
3       Input attributes:     Ani
4                             dnis
5                             Web_DB_Load
6                             Promos_Of_The_Day
7                             Cust_Rec
8                             Home_Phone
9                             Account_Number
10                            Cust_Value
11                            Frustration_Score
12                            Late_Payments_Score
13                            Recent_Purchases
14                            Marketing_VS_Collections
15                            Web_Destinations
16                            Call_Priority
17                            Skill
18                            On_Queue_Promo
19                            Screen_Pop_List
20                            Promo_Hit_List
21      Output attributes:    wrap_up : set ( tuple ( att_name: string,
22                                                    value: string ))
23      Enabling condition:   true
24      Type:            decision
25      Computation:
26          Rules:            if true then wrap_up <- (att_name: "DNIS",
27                                  value : convert-to-string (DNIS))
28                            if true then wrap_up <- (att_name: "ANI",
29                                  value: convert-to-string (ANI))
30                            if true then wrap_up <- (att_name: "skill",
31                                  value: skill)
32                            if web_destinations not empty then wrap_up <-
33                                 (att_name: \"web_destinations",
34                                  value: (convert-to-string
35                                        (web_destinations))
36                            if cust_rec.card_color = "gold" <-
37                                 (att_name:"frustration_score",
38                                  value: convert-to-string
39                                  (frustration_score))
40      Combining policy:          wrap_up_cp //use contributions of all
41                                            rules with true condition
42      Side-effect:               yes
43      Side-effect function:      write_into_archive ( wrap_up )
```

09/251998

FIG. 12

```
1   Module:    get_recent_contacts_for_this_customer

2      Submodule of:    info_about_customer

3      Input attributes:      account_number

4      Output attributes:     recent_contacts : list ( tuple ( date: date,
5                                                      event: event_type,
6                                                      delay_during_contact: int,
7                                                           \\ minutes
8                                                      delay_before_shipment: int
9                                                           \\ days
10                                                     amount: $value ) )
11     Enabling condition:    true

12     Type:                  foreign

13     Computation:           using recent_contacts_db
14                            select date,event,amount
15                            from contact_db
16                            where acct_num = account_number

17     Side-effect:           no
```

FIG. 13

```
1   Module:    get_recent_purchases_for_this_customer

2       Submodule of:    info_about_customer

3       Input attributes:      account_number

4       Output attributes:     recent_purchases : list ( tuple ( date: date,
5                                                        item : 20digit,
6                                                        qty : int,
7                                                        amount : $value ) )
8       Enabling condition:    true

9       Type:              foreign

10      Computation:           using purchase_db
11                             select date,item,qty,amount
12                             from purchases
13                             where acct_num = account_number

14      Side-effect:           no
```

FIG. 14

```
1   Module:    get_account_history_for_this_customer

2       Submodule of:    info_about_customer

3       Input attributes:    account_number

4       Output attributes:   account_history : tuple ( overdue amount:
5                                                          $value,
6                                                    number_days_overdue:
7                                                          int,
8                                                    history: list ( tuple (
9                                                          date: date,
10                                                   item : 20digit,
11                                                   amount : $value ) ) )
12      Enabling condition:  true

13      Type:                foreign

14      Computation:             using account_history_db
15                               select over_amt, num_days,history
16                               from account_history
17                               where acct_num = account_number

18      Side-effect:             no
```

FIG. 15

```
1   Module:   calculate_frustration_score

2       Submodule of:   info_about_customer

3       Input attributes:       recent_contacts

4       Output attributes:      frustration_score : [1..10]

5       Enabling condition:     VAL(recent_contacts)

6       Type:               decision

7       Computation:
8           Rules:              if recent_contacts#1 defined then
9                               frustration_score <-
10                                      (value/50) *
11                                      [(delay_during_contact/2) +
12                                      max(0,delay_before_shipment -
13                                      10)/3]

14                              if recent_contacts#2 defined then
15                              frustration_score <-
16                                      (value/100) *
17                                      [(delay_during_contact/2) +
18                                      max (0,delay_before_shipment -
19                                      10)/3]
20

21          Combining policy: frustration_score_cp //add contributions
22                                                  of true rules and
23                                                  round up, to max
24                                                  of 10
25
26      Side-effect:            no
```

FIG. 16 15/56

```
1  Module:    calculate_net_profit_score
2    Submodule of:    info_about_customer
3    Input attributes:      recent_contacts,
4                           recent_purchases,
5                           account_history,
6                           cust_rec
7    Output attributes:     net_profit_score
8    Enabling condition:    recent_purchases#1.date<=now-60
9    Type:                  decision
10   Computation:
11       Rules:                          if recent_purchases not empty then
12                                       net_profit_score <-
13                                       10% * sum (recent_purchases#i.amount
14                                       where recent_purchases#i.date > now -
15                                       60)
16                                       if recent_contacts not empty then
17                                       net_profit_score <-
18                                       -( 5 * count ( recent_contacts#i
19                                       where recent_contacts#i.type =
20                                       "complaint"))

21                                       if account_history.overdue_amount > 0
22                                       then net_profit_score <-
23                                       - account_history.overdue_amount *
24                                       account_history.number_days_overdue / 30

25                                       if cust_rec.card_color = "platinum" then
26                                       net_profit_score <- 100

27                                       if cust_rec.card_color = "gold" then
28                                       net_profit_score <- 50

29                                       if cust_rec.card_color = "green" then
30                                       net_profit_score <- 10

31                                       if DISABLED(cust_rec) then
32                                       net_profit_score <- 20

33       Combining policy:               net_profit_score_cp //add contributions
34                                                           of rules with true
35                                                           conditions
36
37   Side-effect:           no
```

09/251998

FIG. 17

```
1  Module:    calculate_late_payment_score
2    Submodule of:    info_about_customer
3    Input attributes:      account_history
4    Output attributes:     late_payment_score
5    Enabling condition:    VAL(account_history)
6    Type:              decision
7    Computation:
8        Rules:                    if cust_rec.card_color = "platinum" then
9                                  late_payments_score <-
10                                 (account_history.overdue_amount
11                                 number_of_days_overdue)/100

12                                 if cust_rec.card_color = "gold" then
13                                 late_payments_score <-
14                                 (account_history.overdue_amount *
15                                 number_of days_overdue)/50

16                                 if cust_rec.card_color = "green" then
17                                 late_payments_score <-
18                                 (account_history.overdue_amount *
19                                 number_of days_overdue)/10

20       Combining policy:         late_payment_score_cp //rule with true
21                                                       condition wins;
22                                                       default is 0
23
24   Side-effect:           no
```

FIG. 18

```
1  Module:    calculate_cust_value
2    Submodule of:    info_about_customer
3    Input attributes:      net_profit_score,
4                           late_payments_score,
5                           cust_rec

6    Output attributes:     cust_value
7    Enabling condition:    true
8    Type:              decision
9    Computation:
10       Rules:             if VAL(net_profit_score) then cust_value <-
11                                  (1 - 1/net_profit_score) * 75

12                          if cust_rec.card_color = "platinum" then
13                          cust_value <- 20

14                          if cust_rec.card_color = "gold" then cust_value
15                          <- 10

16                          if cust_rec.card_color = "green" then
17                          cust_value <- 5

18                          if VAL (frustration_score) then cust_value ←
19                          5*frustration_score

20       Combining policy:  calculate_cust_val_cp //Add values of true
21                                                  rules and round up, to
22                                                  max of 100, default is
23                                                  0
24
25   Side-effect:           no
```

FIG. 19

```
1  Module:    calculate_marketing_vs_collections
2    Submodule of:    info_about_customer
3    Input attributes:      cust_value,
4                           late_payments_score
5    Output attributes:     marketing_vs_collections
6    Enabling condition:    late_payments_score > 0
7    Type:              decision
8    Computation:
9        Rules:             if late_payments_score > f(cust_value) then
10                          marketing_vs_collections <- "collect"
11                          // f is function from [1..100] into [1..10],
12                          // it could be linear, i.e., f(cust_value) =
13                          //  cust_value/10
14                          // or it could be shallower in beginning and
15                                steeper
16                          // towards end
17
18
19       Combining policy:  marketing_vs_collection_cp //default is
20                                                      "marketing",
21                                                      any rule
22                                                      with true
23                                                      condition
24                                                      wins
25
26   Side-effect:           no
```

FIG. 20

```
1  Module: Ask_Reason_For_Call
2    Submodule of:   routing_decisions
3    Input attributes:      < none >
4    Output attributes:     IVR_choice
5    Enabling condition:    cust_value < 7 and DNIS not =
6                           "Australia_promotion"
7    Type:              foreign
8    Computation:           x := IVR_dip( question(2) ) ;
9                           if x = 1 then IVR_choice := "dom";
10                          else if x = 2 the IVR_choice := "intl";
11                          else IVR_choice[state] = EXC and
12                          IVR_choice[EXC]=1
13
14     Side-effect:            yes

15     Side-effect-function: IVR_dip( question (2) )
```

FIG. 21

```
1   Module: calculate_business_value_of_call
2       Submodule of:  routing_decisions
3       Input attributes:     IVR_choice,
4                             web_destinations,
5                             frustration_score,
6                             marketing_vs_collections,
7                             late_payments_score,
8                             net_profit_score
9       Output attributes:    business_value_of_call : int
10      Enabling condition:   true
11      Type:                 decision
12      Computation:
13          Rules:
14                            if true then business_value_of_call <-
15                                 (cust_value/50 * net_profit_score)

16                            if true then business_value_of_call <-
17                                 10*frustration_score

18                            if DNIS = "Australia_promtion" then
19                                 business_value_of_call <- 100

20                            if "Australia" in web_destinations[i].regions for
21                                 some i where
22                                 web_destinations[i].date_last_modified > now -
23                                 30
24                                 then business_value_of_call <- 100

25                            if IVR_choice = "intl" then business_value_of_call <-
26                                 50

27                            if marketing_vs_collections = "collect" then
28                                 business_value_of_call <-
29                                 (late_payments_score *
30                                 account_history.overdue_amount)/5

31  Combining policy:     business_value_of_call_cp // Add contributions of
32                                                     rules with true
33                                                     conditions and round up,
34                                                     default is 0
35
36      Side-effect:          no
```

FIG. 22

```
1   Module:  Calculate_send_bonus_check

2      Submodule of:    routing_decisions

3      Input attributes:        cust_rec

4      Output attributes:       send_bonus_check?

5      Enabling condition:      if net_profit_score > 1000
6                               and cust_rec.last_sent_bonus_check < now - 60
7                               and marketing_vs_collections = "market"
8                               OR
9                               if net_profit_score > 500
10                               and frustration_score > 8
11                              and cust_rec.last_sent_bonus_check < now - 60
12                               and marketing_vs_collections = "market"
13

14     Type:              foreign

15     Side-effect:             yes

16          side-effect-function:

17               issue_and_send_check($50,cust_rec.name,cust_rec.address)
```

FIG. 23

```
1.  Module: call_priority

2       Submodule of:   routing_decisions

3       Input attributes:       business_value_of_call
4                               frustration_score

5       Output attributes:      call_priority

6       Enabling condition:     true

7       Type:                   decision

8       Computation:
9         Rules:                if business_value_of_call < 25 then
10                                    call_priority <- 1

11                              if 25 =< business_value_of_call < 100 then
12                                    call_priority <- 2

13                              if 100 =< business_value_of_call < 500 then
14                                    call_priority <- 3

15                              if 500 =< business_value_of_call then
16                                    call_priority <- 4

17                              if frustration_score > 8 then
18                                    call_priority <- 4

19                              if 6 =< frustration_score <= 8 then
20                                    call_priority <- 3

21        Combining policy: call_priority_cp // high value wins with
22                                                default result 2
23
24      Side-effect:            no
```

FIG. 24

```
1  Module: calculate_skill
2     Submodule of:   routing_decisions
3     Input attributes:          business_value_of_call
4                                marketing_vs_collections
5                                IVR_choice
6                                DNIS
7                                web_destinations
8     Output attributes:         skill
9     Enabling condition:        true
10    Type:           decision
11    Computation:
12       Rules:       if marketing_vs_collections = "collections"
13                       then skill <- ["collections", infinity]
14                    if business_value_of_call > 100
15                       then skill <- ["high_tier", 40]
16                    if DNIS = "australia_promotion" then
17                       skill <- ["australia_promo", infinity]
18                    if "Australia" in web_destinations[i].regions
19                       for some i where web_destinations[i].date_last_modified >
20                       now - 30 then
21                       skill <- ["australia_promo", 20]
22                    if cust_rec.estimated_income_bracket = ">100K-150K" then
23                       skill <- ["australia_promo", 25]
24
25                    if cust_rec.estimated_income_bracket = ">150K" then
26                       skill <- ["australia_promo", 35]
27
28                    if IVR_choice = "dom" then skill <- ["norm_tier_dom",30]
29
30                    if IVR_choice = "intl" then skill <- ["norm_tier_intl",30]
31
32                    if "US" in web_destinations[i].regions for some
33                       i where web_destinations[i].date_last_modified >
34                       now - 30 then
35                       skill <- ["norm_tier_dom", 20]
36
37                    if "US" not in web_destinations[i].regions for
38                       some i where web_destinations[i].date_last_modified > now -
39                       30 then
40                       skill <- ["norm_tier_intl", 20]
41       Combining policy: skill_cp //weighted sum policy, and ties are
42                                broken by ordering "collections",
43                                "australia_promo", "high_tier",
44                                "low_tier_intl", "low_tier_dom",
45                                default is ⊥
46
47    Side-effect:  no
```

09/251,998

FIG. 25

```
1  Module:  calculate_on_queue_promo

2     Submodule of:  routing_decisions

3     Input attributes:        promo_hit_list

4     Output attributes:       on_queue_promo

5     Enabling condition:      DISABLE if business_value > 100 or

6  frustration_score > 5

7     Type:            decision

8     Computation:
9        Rules:                if 60 < ACD.expected_wait_time(skill)
10                                  then on_queue_promo <-
11                                  promo_hit_list[#1]

12                            if business_value_of_call < 30
13                                  then on_queue_promo <- promo_hit_list[#1]

14       Combining policy:     on-queue-promo-cp // first true wins, default
15                                                  is 0
16
17    Side-effect:             no
```

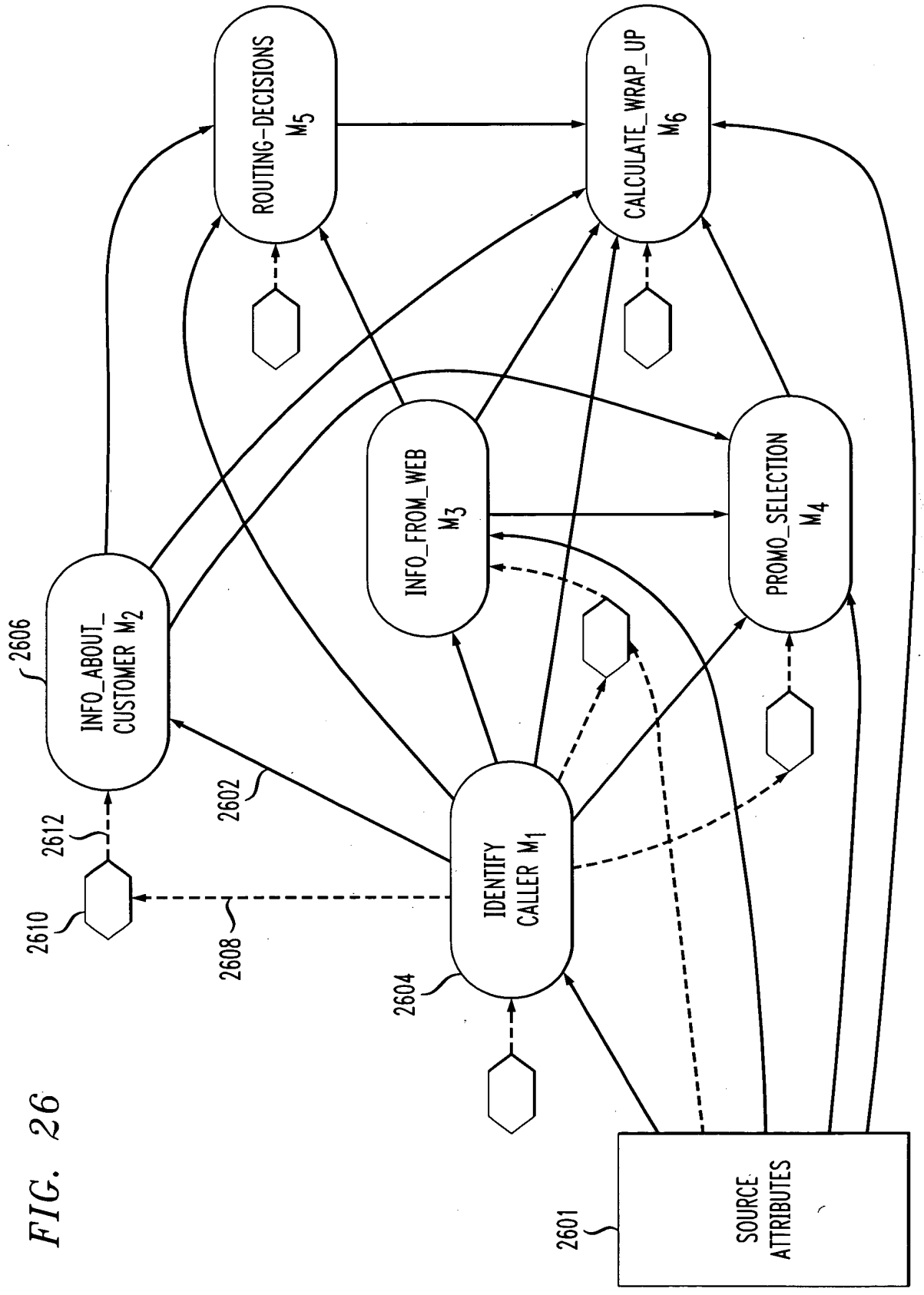


FIG. 26

FIG. 27 26/56

$\dfrac{\sigma \vdash e:t}{\sigma \vdash \text{value}(e):\ bool}$	VALUE
$\dfrac{\sigma \vdash f{:}AM{:}t_1 x \ldots x t_n \rightarrow t,\ \sigma \vdash e_1:t_1, \ldots \sigma \vdash e_n:t_n}{\sigma \vdash \text{Apply}(<f,e_1,\ldots,e_n>):t}$	APPLY
$\dfrac{\sigma \vdash e_1:t_1, \ldots \sigma \vdash e_n:t_n}{\sigma \vdash <e_1,\ldots,e_n>:<a_1:t_1,\ldots,a_n:t_n>}$	TUPLING
$\dfrac{\sigma \vdash e_1:t, \ldots, \sigma \vdash e_n:t}{\sigma \vdash \{e_1,\ldots,e_n\}:\{t\}}$	BAGGING
$\dfrac{\sigma \vdash e_1:t, \ldots, \sigma \vdash e_n:t}{\sigma \vdash [e_1,\ldots,e_n]:[t]}$	LISTING
$\dfrac{\sigma \vdash e:\{t\}}{\sigma \vdash \text{unitval}(e):t}$	UNITVAL
$\dfrac{\sigma \vdash <a_1:t_1,\ldots,a_n:t_n>}{\sigma \vdash e.a_i:t_i}$	PROJECTION ON TUPLES
$\dfrac{\sigma \vdash e:[t]}{\sigma \vdash e\#i:t}$	PROJECTION ON LISTS
$\dfrac{\sigma \vdash e_1:[t_1], \sigma \vdash e_2:t_2}{\sigma \vdash \text{factor}(e_1,e_2):[<\text{f\_}a:t_1,\text{s\_}a:t_2>]}$	FACTOR (ON LISTS)
$\dfrac{\sigma \vdash e_1:\{t_1\}, \sigma \vdash e_2:t_2}{\sigma \vdash \text{factor}(e_1,e_2):\{<\text{f\_}a:t_1,\text{s\_}a:t_2>\}}$	FACTOR (ON BAGS)
$\dfrac{\sigma \vdash f:t_1 \rightarrow t, \sigma \vdash S:[t_1]}{\sigma \vdash \text{map}(f)(S):[t]}$	MAP (ON LISTS)
$\dfrac{\sigma \vdash f:t_1 \rightarrow t, \sigma \vdash S:\{t_1\}}{\sigma \vdash \text{map}(f)(S):\{t\}}$	MAP (ON BAGS)
$\dfrac{\sigma \vdash id_\theta:t, \sigma \vdash \theta:txt \rightarrow t, \sigma \vdash S:\{t\}}{\sigma \vdash \text{collect}(id_\theta,\theta)(S):t}$	COLLECT (ON BAGS)
$\dfrac{\sigma \vdash id_\theta:t, \sigma \vdash \theta:txt \rightarrow t, \sigma \vdash S:[t]}{\sigma \vdash \text{collect}(id_\theta,\theta)(S):t}$	COLLECT (ON LISTS)

FIG. 28

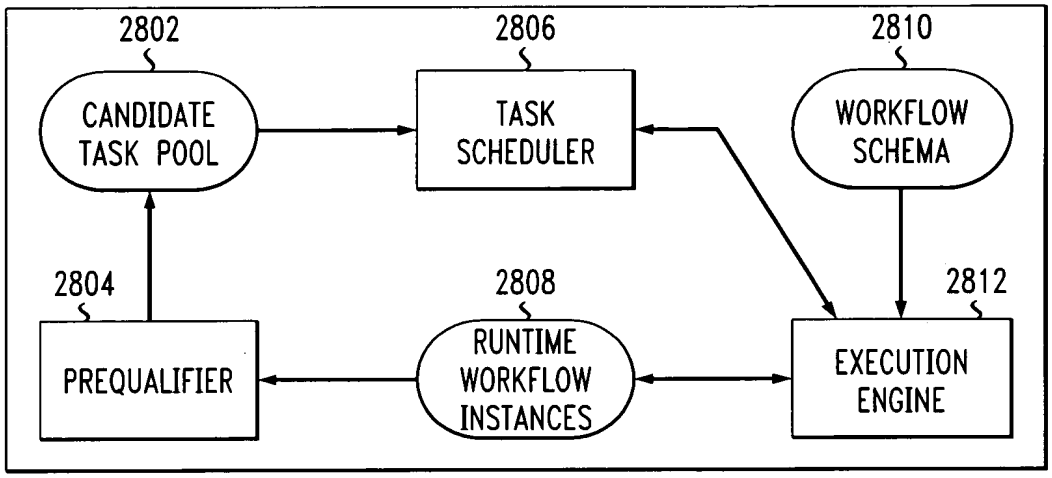


FIG. 29

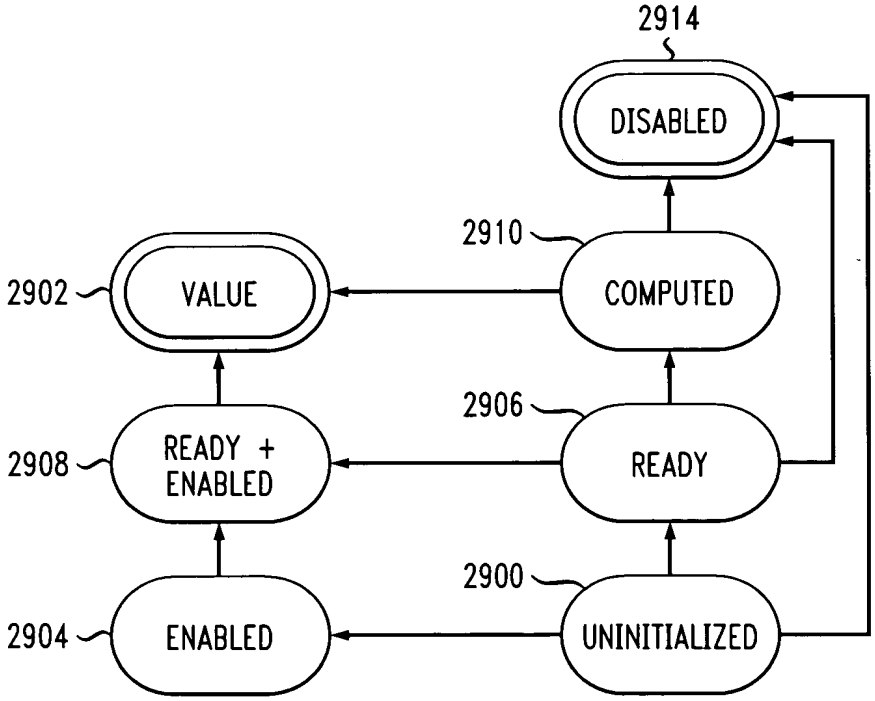


FIG. 30

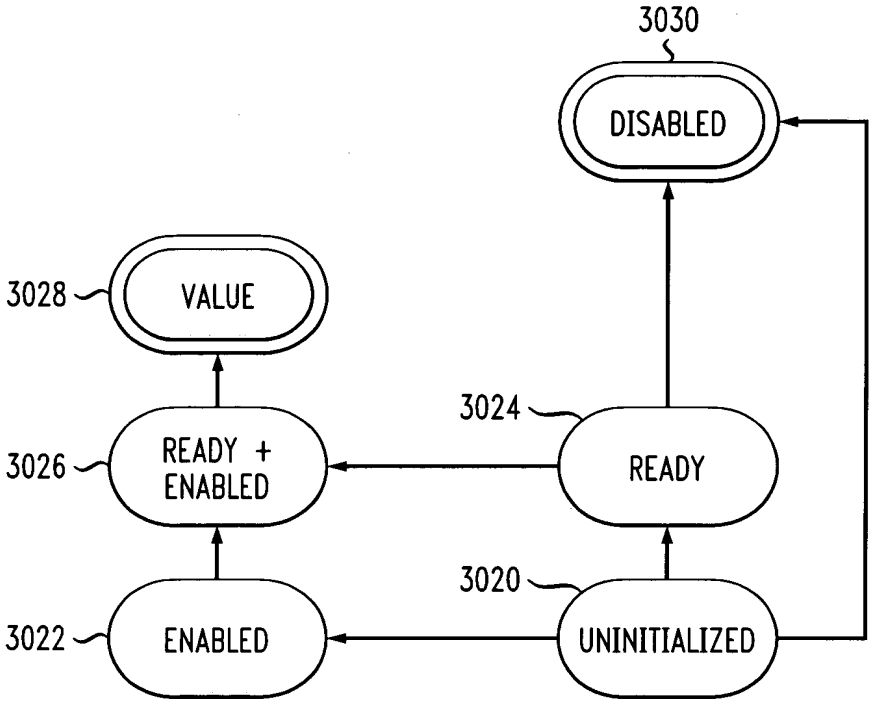
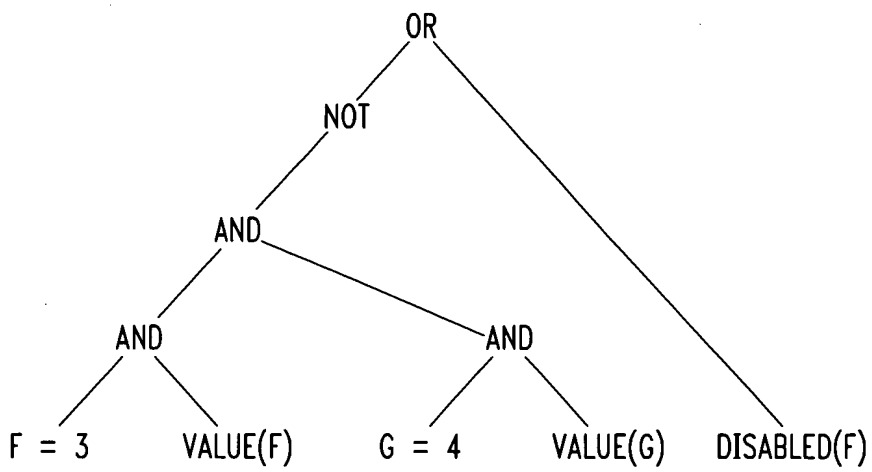


FIG. 31



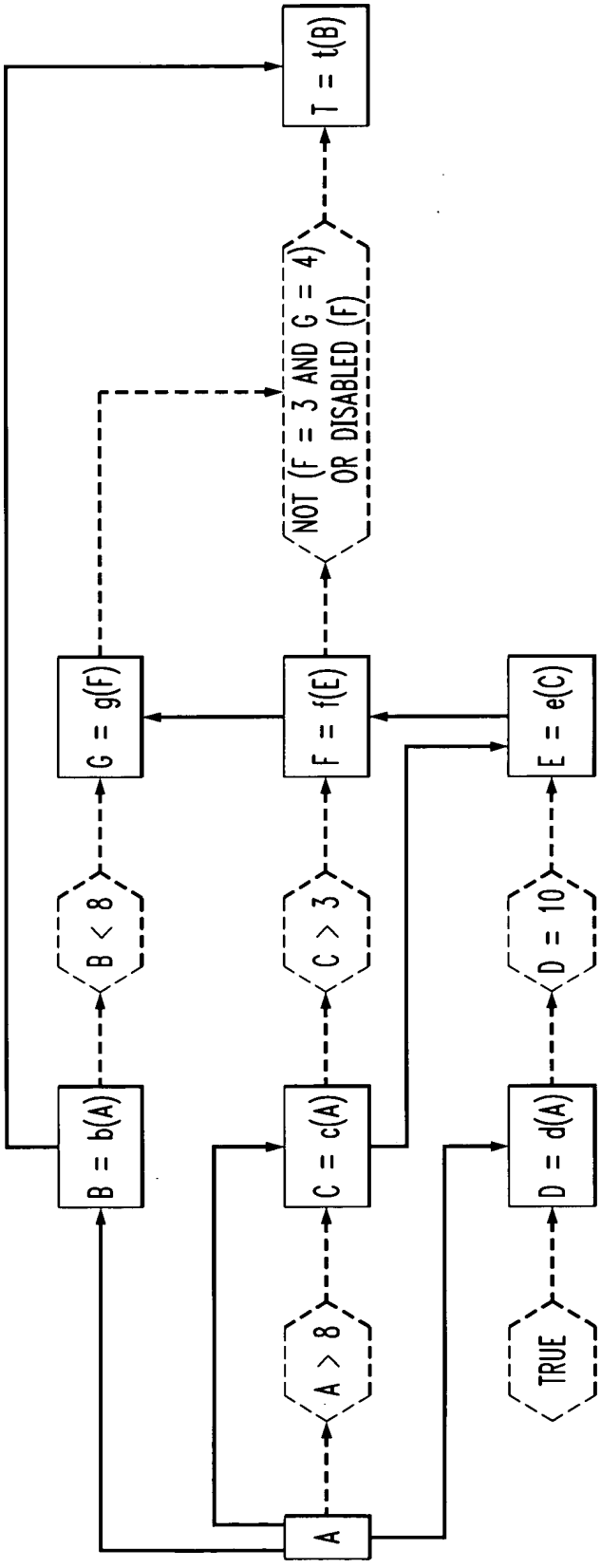


FIG. 32

FIG. 33

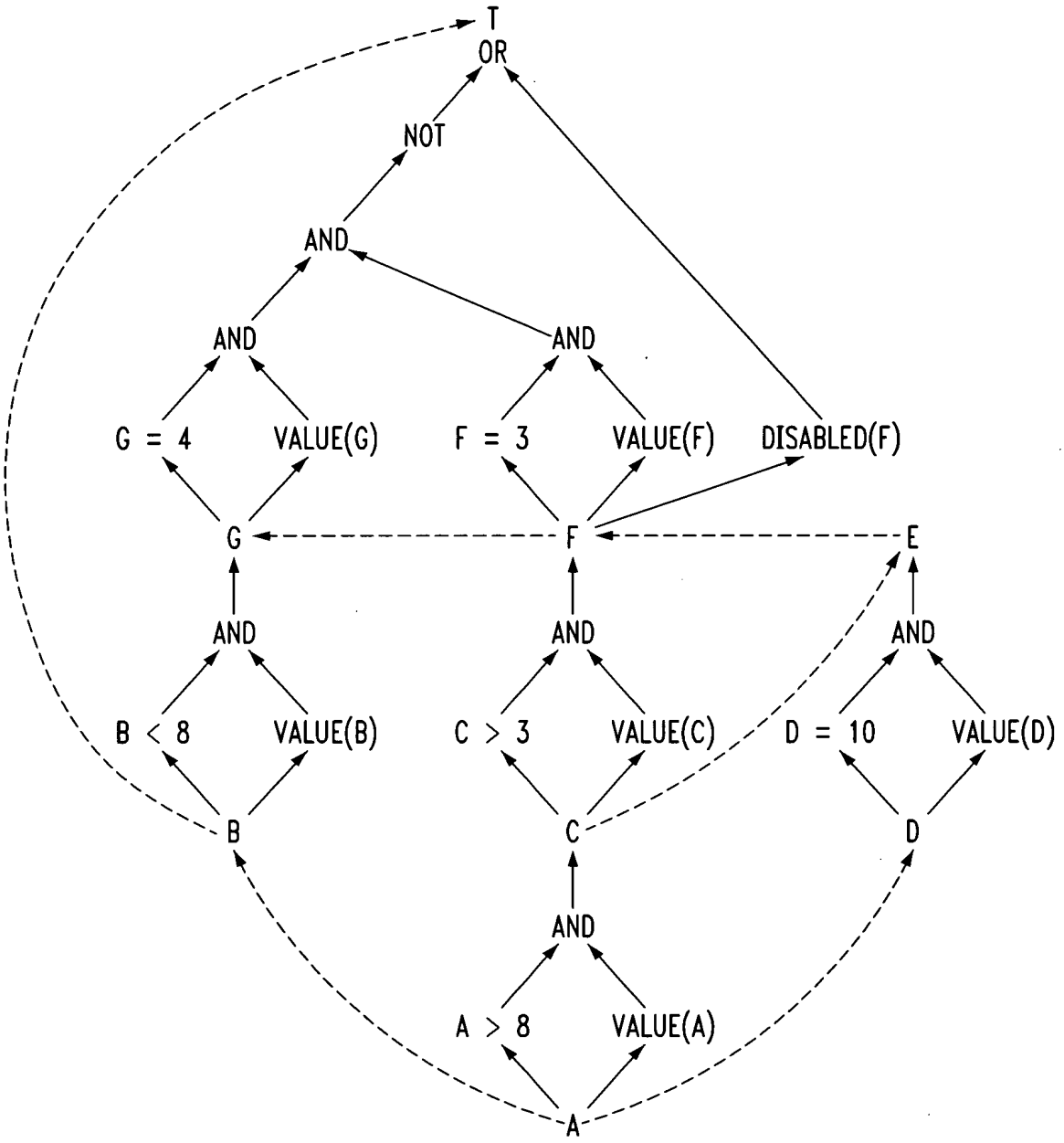


FIG. 34A

Global variables:

These variables are global to the whole execution of workflow instance 3402
- *G*: a dependency graph
- *S*: set of source attribute nodes of *G*
- *T*: set of target attribute nodes of *G*
- σ []: array of attribute states
- μ []: array of attribute values
- α []: array of three valued logic values (*true, false unknown*)
- *HIDDEN_EDGE*: set of hidden edges of *G*.
- *HIDDEN_ATT*: set of hidden attribute nodes of *G*.

Notations: 3404
- σ [*A*]: element of array σ [] that corresponds to the attribute node *A* in *G*
- μ [*A*]: element of array μ [] that corresponds to the attribute node *A* in *G*
- α [*p*]: element of array α [] that corresponds to the condition node *p* in *G*

FIG. 34A (cont)

```
Initialization phase:
    procedure Init:
    Input:
        g: a dependency graph:
        So: source nodes in g
        Te: terminal nodes in g
    body:
    BEGIN init
        G:=g ; S: = So: T: = Te;
        /*Initialization of the states and values of attributes nodes */
        FOR all the attribute nodes A in G DO
            IF A ∈ S  /* A is a source node */
              THEN σ[A] := READY + ENABLED
              ELSE σ[A] := UNITIALIZED;
        μ[A] :=NULL;
        END FOR
        /* Initialization of α-values of condition nodes */
        FOR all the condition nodes p in G DO
          α[A] := unknown;
        END FOR
        */ Initialization of the set of hidden edges and hidden nodes */
        HIDDEN_EDGE:=∅; HIDDEN_ATT:=∅
    END init
```

3406
3408
3410
3412

FIG. 34B

```
Increment
    Input:
        A : an attribute in C.
        v : a value for A.                                  ~ 3416
    body:
        BEGIN increment
      μ [A] : = v;                                          ~ 3418
      IF σ [A] = READY
           THEN propagate_att_change(A, COMPUTED)           ~ 3420
      IF σ [A] = READY+ENABLED
           THEN propagate_att_change(A, VALUE)
END Increment                                               ~ 3414

propagate_att_change                                        ~ 3422
      Input:
          B : an attribute in C.
          σ : a state for B                                 ~ 3424
      body:
    /* Set state for B*1
    IF ((σ[B] = ENABLED) AND ( σ = READY)) OR (σ[B] = READY) AND (σ =ENABLED))
          THEN σ [B] := READY+ENABLED
          ELSE σ [B] := σ;                                  ~ 3426
```

FIG. 34B (cont)

```
/* push relevant information to the affected successor nodes */
CASE : σ [B] ∈ {VALUE, COMPUTED} /* The value of B is computed */
  /* try to evaluate predicate nodes that are using the value of B */
  FOR each condition node p of the form pred(t_1, ··· , t_n) such that (B,p) ∈ G DO ~ 3432
        IF (B,p) ∉ HIDDEN_EDGE ~ 3434
              THEN
                 Hide_edge ((B,p)); ~ 3436
                 IF Eval(p) ≠ unknown THEN α [p] :=Eval(p); propagate_cond_change(p); ~ 3438
  END FOR                                                                              } 3430
/* check if the attributes nodes that have B as input parameters are READY */
FOR each attribute node C such that (B, C) ∈ G DO
IF σ [B] =VALUE THEN
        IF (B, C) ∉ HIDDEN_EDGE
              THEN                                                                   ~ 3440
                 Hide_edge((B,C));
                 IF there exists no attribute node D such that (D, C) ∉ HIDDEN_ EDGE
                       THEN propagate_att_change (C READY);
END FOR
CASE : σ [B] = ENABLED
/* evaluates condition nodes of the form VALUE (B) and DISABLED (B) */
FOR each condition node p of the form VALUE (B) or DISABLED (B) such that (B,p) ∈ G DO
        IF (B,p) ∈ HIDDEN_ EDGE                                                      ~ 3442
              THEN
                 Hide_edge((B,p));
                 IF p is of the form VALUE (A) THEN α[p] := true ELSE α[p]: = false;
                 propagate_cond_change(p);
```

3422

3428

FIG. 34C

```
    END FOR
CASE: σ [B] = DISABLED
    /* evaluate condition nodes of the form VALUE (B) and DISABLED (B) */
    FOR each condition node p of the form VALUE (B) or DISABLED (B) such that (B,p) ∈ G DO
      IF (B,p) ∉ HIDDEN_EDGE
          THEN
            Hide_edge ((B,p));
            IF p is of the form VALUE (A) THEN α[p] := false ELSE α[p] := true;
            propagate_cond_change(p);
    END FOR
    /* check if the attribute nodes that have B as input parameters are READY */
    FOR each attribute node C such that (B,C) ∈ G DO
      IF (B,C) ∉ HIDDEN_EDGE
          THEN
            Hide_edge((B,C));
            IF there are no more attribute nodes D such that (D,C) ∉ HIDDEN_EDGE
               THEN propagate_att_change (C,READY);
    END FOR
    /* If the attribute is stable then hide the attribute */
      IF (σ[B] ∈ {DISABLED, VALUE}) THEN Hide_node(B);
    END propagate_att_change
```

3444 3446 3428 3422 3448

Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560

FIG. 34C (cont)

propagate_cond_change

Input:
p: a condition node in G.
body:
BEGIN propagate_cond_change
let n be the successor of p in G ~ 3452

IF $(p,n) \notin$ *HIDDEN_EDGE*
THEN
Hide_edge $((p,n))$; ~ 3456
CASE: n is OR condition node
IF ($\alpha[p]$ = *true*) THEN $\alpha[n]$: = *true*; propagate_cond_change(n); END IF; ~ 3460
IF $\alpha[p]$ = *false* AND for each condition node p' where $(p',n) \in G$, $(p',n) \in$ HIDDEN_EDGE THEN α [n] : = *false*; propagate_cond_change(n);END IF; ~ 3462
~ 3458
CASE: n is a AND node
IF ($\alpha[p]$ = *false*) THEN α [n] := *false*; propagate_cond_change(n);END IF; ~ 3466
IF $\alpha[p]$ = *TRUE* AND for each condition node p' where $(p',n) \in G$, $(p',n) \in$ HIDDEN_EDGE THEN α [n] : = *TRUE*; propagate_cond_change(n);END IF; ~ 3468
~ 3464
CASE: n is NOT node
$\alpha[n]$: = $\neg(\alpha[p])$; propagate_cond_change(n); ~ 3470
~ 3454
CASE: n is an attribute node
IF ($\alpha[p]$ = *true*)
THEN propagate_att_change(n,ENABLED)
ELSE propagate_att_change(n,DISABLED); ~ 3472
END propagate_cond_change
~ 3450

Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560

FIG. 34D

```
Hide_edge
  Input
      (n,n') : an edge in G.
body
BEGIN Hide_edge
HIDDEN_EDGE: = HIDDEN_EDGE U {(n,n')];

IF (there are no more edges (n,n'') ∈ G such that (n,n'') ∉ HIDDEN_EDGE
      THEN Hide_node(n)
END Hide_edge
```

3474

```
Hide_node
   Input
       n : a node in g.
   body
   BEGIN Hide_node
   HIDDEN_ATT: = HIDDEN_ATT U {n}

   FOR each edge (n',n) ∈ g such that (n',n) ∉ HIDDEN_EDGE) DO
      Hide_edge (n',n)
   END FOR
END Hide_node
```

3476

FIG. 35A

Global variables:

These variables are global to the whole execution of workflow instance

G : a dependency graph
S : set of attribute nodes of *G* /* this set contains the source nodes */
T : set of attribute nodes of *G* /* this set contains target nodes */
σ [] : array of attribute states
α [] : array of three valued logic values (*true, false unknown*)
HIDDEN_EDGE : set of edges of *G*.
HIDDEN_ATT : set of attribute nodes of *G*.

3504

T_N[][] : Matrix of integers that associates an integer value to each pair (*p*,*A*) where *p* is a condition node and *A* is an attribute node in *G*
/* *T_N*[*p*][*A*] = 0 means that the attribute *A* is True_necessary for the condition node *p*/

F_N[][] : Matrix of integers that associates an integer value to each pair (*p*,*A*) where *p* is a condition node and *A* is an attribute node in *G*
/**F_N*[*p*][*A*] = 0 means that the attribute *A* is False_necessary for the condition node *p*/

V_N [][] : Matrix of integers associates an integer value to each pair (*B*,*A*) where *B* and *A* are attribute nodes in *G*
/**V_N*[*B*][*A*] = 0 means that the attribute *A* is Value_necessary for the attribute note *B**/

3502

FIG. 35A
(cont.)

$S\_N[][]$: Matrix of integers associates an integer value to each pair (B,A) where
B and A are attribute nodes in G
/*$S\_N[B][A]$ = 0 means that the attribute A is Stable_necessary for the attribute node B */

$N[]$: Array of boolean
$N[A]$ = *true* means that the attribute A is computed as necessary/*
$N[A]$ = *false* means that the attribute A is not computed as necessary*/

3502

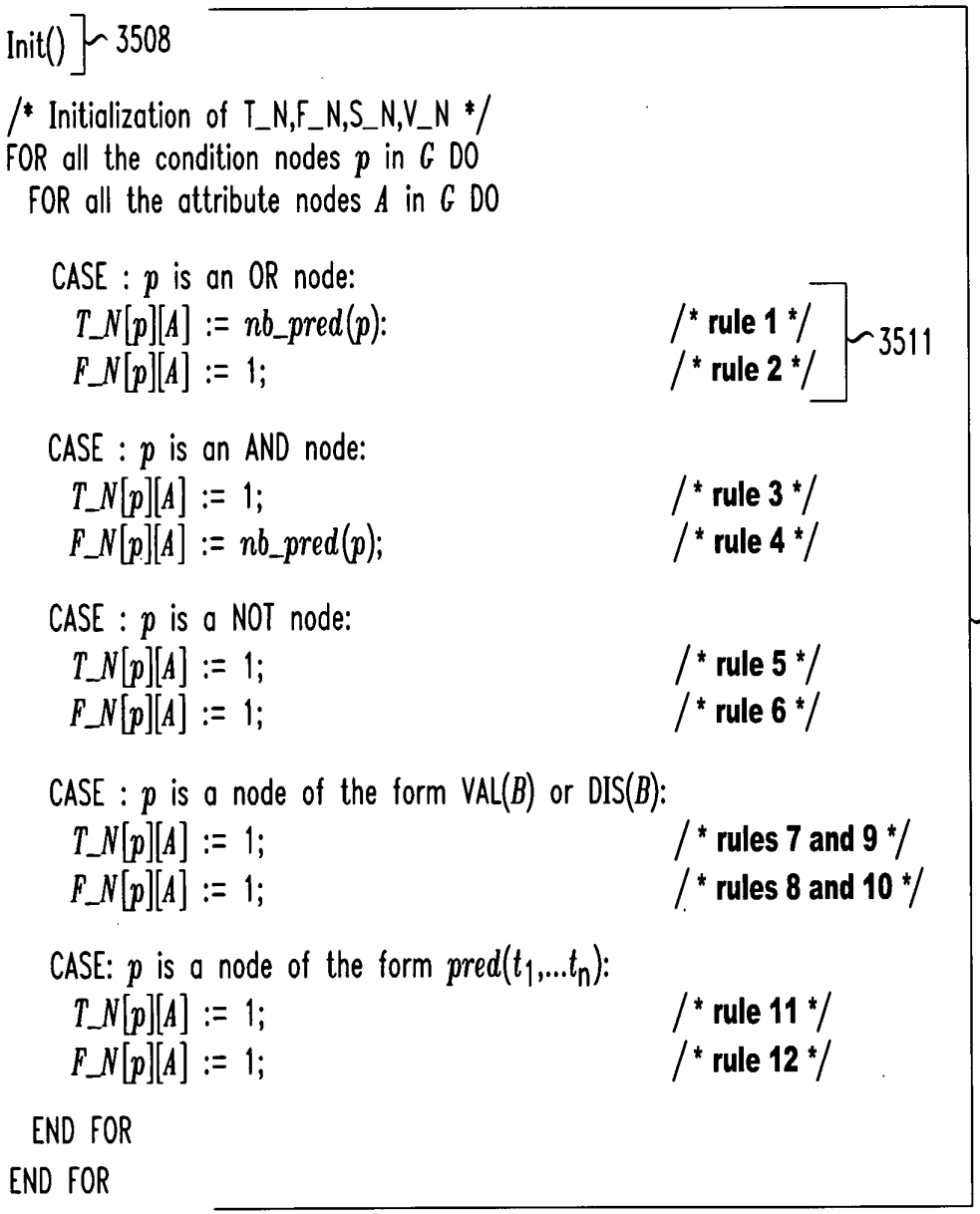
Notations:

$nb\_pred(p)$: number of predecessors of p in G

Initialization phase:
procedure Init :
Input:
g : a dependency graph:
So : source nodes in g
Te : terminal nodes in g
body:
BEGIN N_init

3506

Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560



3506

FIG. 35B

Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560

FIG. 35B
(cont.)

```
FOR all the attributes nodes A in G DO
  FOR all the attribute nodes B in G DO
    S_N[A][B] := 1; V_N[A][B] := 1
  END FOR
END FOR
```
3512

3506

```
FOR all the attributes nodes A in G DO
  N[A] := false
END FOR
```
3513

```
END N_init
```

N-Increment

Input:

A : an attribute in *G*.

v : a value for *A*.

Variables/* Global to one execution of the increment phase (for one execution step) */

*prev_*E: set of attribute nodes
/* used to store the nodes that were READY+ENABLED or ENABLED (in a previous execution of N-increment) */
3518

prev_HIDDEN_EDGE :/* set of edges*/
used to store the edges that were previously hidden (in the previous steps) */

3526

prev_T_N: set of pairs (*p*,*A*) where *p* is a condition node and *A* is an attribute node
/* used to denote the elements of *T_N*[*p*][*A*] that were set to 0 in a previous execution of N-increment*/

prev_F_N: set of pairs (*p*,*A*) where *p* is a condition node and *A* is an attribute node
/* used to denote the elements of *F_N*[*p*][*A*] that were set to 0 in a previous execution of N-increment*/

3518

Δ_E: set of attribute nodes
/* used to store the new ENABLED or READY+ENABLED attribute nodes that were neither ENABLED nor READY+ENABLED in the previous steps. */

Δ_*HIDDEN_EDGE*: set of edges
/* used to store the new hidden edges */

3520

new_V_N : set of pair (*A*,*A*) where *A* is an attribute node
/* used to store the positions of elements of *V_N*[][] whose new value is zero due to case 1 */

new_S_N : set of pair (*B*,*A*) where *B* and *A* are attribute nodes
/* used to store the positions of elements of *S_N*[][] whose new value is zero due to case 2 */

new_T_N : set of pair (p,*A*) where *p* is a predicate node and *A* is an attribute node
/* used to store the positions of elements of *T_N*[][] whose new value is zero due to some new hidden edges (case 3) */

new_F_N : set of pair (p,*A*) where *p* is a predicate node and *A* is an attribute node
/* used to store the positions of elements of *F_N*[][] whose new value is zero due to some new hidden edges (case 4) */

3522

3526

FIG. 35C

FIG. 35D

body:
BEGIN N_increment

/* preparation step: */
prev_HIDDEN_EDGE := *HIDDEN_EDGE*;
prev_E := {*A*|*A* is an attribute node in *G* and σ [*A*] $\in$ {READY+ENABLED, ENABLED}} ~ 3524

Increment(*A*,*v*) ~ 3526

/* Instigation step : Compute new necessary properties according to the instigation cases*/ ~ 3528

Case 1:
Δ_E := {*A*|*A* is an attribute node in *G* and σ [*A*] $\in$ {READY+ENABLED, ENABLED} and *A* $\notin$ *prev_E*} ~ 3530
new_V_N := $\emptyset$;
FOR each attribute node *A* in Δ_E DO
V_N[*A*][*A*] := 0; *new_V_N* := *new_V_N* U {(*A*,*A*)}/* a node is value_necessary for itself*/
END FOR

Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560

FIG. 35D (cont.)

Case 2:

new_S_N := ∅;
FOR each attribute node *B* in Δ_E DO
  FOR each attribute node in *A* in *G* such that σ [*A*] ∈ {READY+ENABLED, ENABLED} DO
    IF *V_N*[*B*][*A*] = 0 and *S_N*[*B*][*A*] = 1
      THEN *S_N*[*B*][*A*] = 0; *new_S_N*:= *new_S_N* U {(*B*,*A*)} /* **rule (13)***/
  END FOR
  END FOR

3532

Δ_*HIDDEN_EDGE*:= *HIDDEN_EDGE*-*prev_HIDDEN_EDGE*
prev_T_N := {(*p*,*A*) | *T_N*[*p*][*A*] =0 }
prev_F_N := {(*p*,*A*) | *F_N*[*p*][*A*] =0 }
new_T_N := ∅;
new_F_N := ∅;

3534

FOR all edges (*n*,*p*) ∈ Δ_*HIDDEN_EDGE* such that *p* ∉ *HIDDEN_ATT* and *p* is a condition node DO
  FOR all attribute nodes *A* such that σ (*A*) ∉ {COMPUTED, VALUE, DISABLED} DO
  CASE: 3
  CASE : *p* is an OR node:
    IF (*n*,*A*) ∉ *prev_T_N*
    THEN
      T_N[*p*][*A*] := *T_N*[*p*][*A*]-1; /* **rule (1)***/
      IF *T_N*[*p*][*A*] = 0 THEN *new_T_N* := *new_T_N* U {(*p*,*A*)}

3536

3528

8661 SC) bQ

FIG. 35E

```
CASE: 4
CASE : p is an AND node :
IF (n,A) ∉ prev_F_N /* same reasoning as for OR nodes but with rule 4*/
THEN
  F_N[p][A] := F_N[p][A] -1;          /* rule (4)*/
  IF F_N[p][A] = 0 THEN new_F_N := new_F_N U {(p,A)}
END FOR
END FOR
```

3538

3528

```
/* Propagation step */
    New_propagate(new_V_N, new_S_N, new_T_N, new_F_N)        ~ 3540
END N_Increment

New_propagate
  Input:
    new_V_N : set of pairs (A,A) where A is an attribute node
    new_S_N : set of pairs (B,A) where B and A are attribute nodes
    new_T_N : set of pairs (p,A) where p is a condition node in G and A is an attribute
    node
    new_F_N : set of pairs (p,A) where p is a condition node in G and A is an attribute
    node
  body:
  FOR each pair (A,A) in new_V_N DO                                          ~ 3542
    propagate_V_N(A,A)
    FOR each attribute node B such that (A,B) ∈ G and (A,B) ∉ HIDDEN_EDGE
      V_N[B][A] := 0; propagate_V_N(B,A)/*       rule (16) */                ~ 3546
    END FOR
  END FOR
  FOR each pair (B,A) in new_S_N DO                                          ~ 3544
    propagate_S_N(B,A)
  END FOR
    FOR each pair (p,A) in new_T_N DO
      propagate_T_N(p,A)
  END FOR
  FOR each pair (p,A) in new_F_N DO
      Propagate_F_N(p,A)
  END FOR
END N-propagate
```

FIG. 35E (cont.)

FIG. 35F

```
propagate_V_N
  Input :
    B : an attribute node in G.
    A : an attribute node in G./* A in newly Value_necessary for B */
  body:
  IF σ [B] = ENABLED and S_N[B][A] = 1
     THEN S_N[B][A] = 0; propagate_S_N(B,A)          /*rule (13)*/   ~ 3550
ELSE let p be the condition node such that (p,B) ∈ G.
      IF F_N[p][A]=0 and S_N[B][A] = 1
        THEN S_N[B][A] = 0; propagate_S_N(B,A)       /*rule (14)*/   ~ 3552
END IF                                                               ~ 3548
FOR each condition node p of the form pred(t1...,tn)
      such that (B,p) ∈ g and (B,p) ∉ HIDDEN_EDGE DO
IF T_N[p][A] = 1
   THEN T_N[p][A] : = 0;propagate_T_N(p,A)           /*rule (11)*/   ~ 3554
IF F_N[p][A] = 1                                                     ~ 3542
  THEN F_N[p][A] : =0;propagate_F_N(p,A)             /*rule (12)*/   ~ 3556
  END FOR
END propagate_V_N

propagate_S_N
  Input:
     B : an attribute node in G.
     A : an attribute node in G./* A is newly Stable_necessary for B*/
  body:
  FOR each attribute node C such that (B,C) ∈ g and (B,C) ∉ HIDDEN_EDGE DO
     IF V_N[C][A] = 1 THEN V_N[C][A] = 0;propagate_V_N(C,A) /* Rule 17 */   ~ 3560
  END FOR                                                                   ~ 3558
  IF B ∈ T THEN N[A] : =true                                                ~ 3562
END propagate_S_N
```


Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560

```
propagate_F_N
 Input:
  p : a condition node in G.
  A : an attribute node in G./* A is newly False_necessary for p*/
 body:
  let n be the successor of p in G
  IF (p,n) ∈ HIDDEN_EDGE
      THEN
       CASE : n is an OR or AND node
         IF F_N[n][A] > 0
            THEN
            F_N[n][A] := F_N[n][A] - 1;        /*rules (2) and (4)*/
            IF F_N[n][A] = 0 THEN propagate F_N (n,A)
         CASE : n is a NOT node
          IF T_N[n][A] = 1 THEN T_N[n][A] : =0;propagate_T_N(n,A) /*rule (6)*/
         CASE : n is an attribute node
          IF (T_N[p][A] = 0 or V_N[n][A] = 0 and S_N[n][A] = 1
              THEN S_N[n][A] = 0;propagate_S_N(n,A)     /*rules (14) and (15)*/
           FOR each condition node p' of the form VALUE (n)
               such that (n,p') ∈ g and (n,p') ∉ HIDDEN_EDGE DO
          IF F_N[p'][A] = 1 THEN F_N[p'][A] : =0;propagate_F_N(p',A) /*rule (8)*/
         END FOR
         FOR each condition node p' of the form DISABLED (n)
             such that (n,p') ∈ G AND (n,p') ∉ HIDDEN_EDGE DO
       IF T_N[p'][A] = 1 THEN (T_N[p'][A] :=0;propagate_T_N(p',A) /*rule (10)*/
      END FOR
END propagate_F_N
```

3564

3542

FIG. 35F
(cont.)

Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560

```
propagate_T_N
 Input:
  p : a condition node in G.
  A : an attribute node in G/* A is newly True_necessary for p */
 body:
  let n be the successor of p in G
  IF (p,n) ∉ HIDDEN_EDGE
     THEN
      CASE : n is an OR or AND node
       IF T_N[n][A]>0
          THEN
          T_N[n][A] := T_N[n][A] - 1; /*rules (1) and (3)*/
          IF T_N[n][A] = 0 THEN propagate_T_N(n,A)
      CASE : n is a NOT node
       IF F_N[n][A] = 1 THEN F_N[n][A] : = 0; propagate_F_N(n,A) /*rule (5) */
      CASE : n is an attribute node
       IF F_N[p][A] = 0 and S_N[n][A] = 1
          THEN S_N[n][A] =0;propagate_S_N(n,A) /*rule (15)*/
       FOR each condition node p' of the form VALUE(n)
            such that (n,p') ∈ G and (n,p') ∉ HIDDEN_EDGE DO
          IF T_N[n][A] = 1 THEN
                T_N[p'][A] : = 0;propagate_T_N(p',A)     /*rule (8)*/
       END FOR
       FOR each condition node p' of the for DISABLED (n)
          Such that (n,p') ∈ G and (n,p') ∉ HIDDEN_EDGE DO
        IF F_N[n][A] = 1 THEN
        F_N[p'][A] : =0;propagate_F_N(p',A)       /*rule (9)*/
       END FOR
END propagate_T_N
```

3566

3542

FIG. 35G

Hull 5-4-1-4
Serial No.: 09/251,998
Ryan, Mason & Lewis, LLP; R. J. Mauri (203) 255-6560

FIG. 36

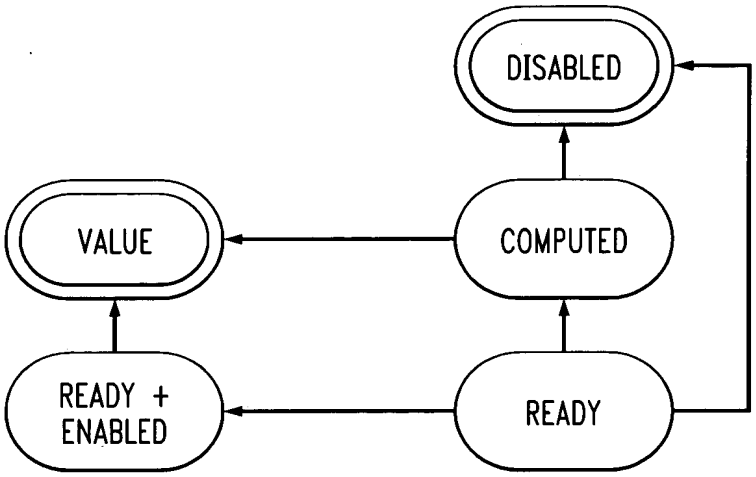


FIG. 37

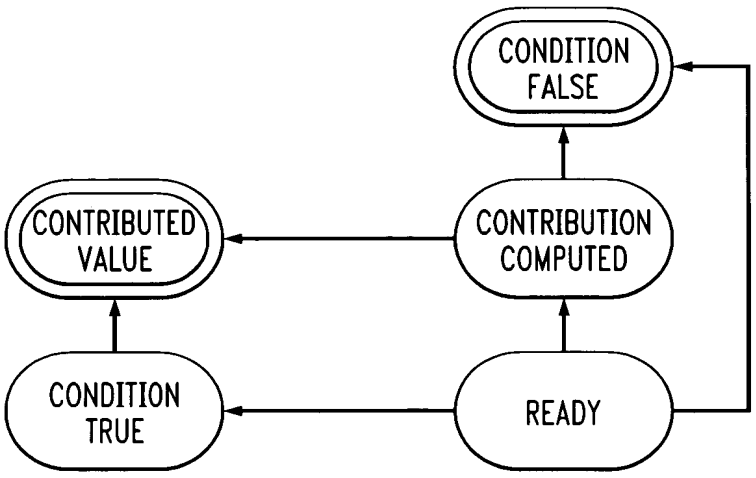


FIG. 38

	A	B	C	D	E
1	source		get_recent_ contacts_... (node 504)	get_recent_ purchases_... (node 508)	get_account_ history_... (node 512)
2			foreign module	foreign module	foreign module
3	cust_rec	account_ number	recent_ contacts	recent_ purchases	account_ history
4	<"John Doe", "101 Ash, LA", "gold", FALSE, SV ...>	421135 SV	NS	NS	NS
5			ENABLED + READY	ENABLED + READY	ENABLED + READY
6					
7					
8					
9					
10					

09/25/1998

FIG. 38 (cont) 52/56

F	G	H	I	J
calculate_ frustration_ score (node 516)	calculate_ net_profit_ score (node 520)	calculate_ late_payments_ score (node 524)	calculate cust_value (node 528)	calculate_ marketing_vs_ collections (node 532)
"add contribs. of true rules and round up, to max of 10"	"add contribs. of true rules"	"true rule wins; default is 0"	"add contribs. of true rules and round up, to max of 100"	"any true rule gives collect; default is marketing"
frustration_ score	net_profit_ score	late_ payment_ score	cust_ value	marketing_ vs_ collections
NS	NS	NS	NS	NS
READY	READY	READY	ENABLED + READY	READY
READY	READY	⊥	READY	"collect" C-C
READY	READY	condition true	⊥	
	READY	⊥	10 C-V	
	⊥		⊥	
	50 C-V		READY	

09/251,998

FIG. 39

	A	B	C	D	E
1	source		get_recent_ contacts_... (node 504)	get_recent_ purchases_... (node 508)	get_account_ history_... (node 512)
2			foreign module	foreign module	foreign module
3	cust_rec	account_ number	recent_ contacts	recent_ purchases	account_ history
4	<"John Doe", "101 Ash, LA", "gold", FALSE, SV ...>	421135 SV	NS	[<8-10-98, coat, 1, $50> <6-15-98, hat, SV 1, $20>]	<10, 45, [<9-18 -98 PAY, $40> <8-10-98, SV ORDER, $50>]
5			ENABLED + READY	VALUE	VALUE
6					
7					
8					
9					
10					

FIG. 39 (cont) 54/56

F	G	H	I	J
calculate_ frustration_ score (node 516)	calculate_ net_profit_ score (node 520)	calculate_ late_payments_ score (node 524)	calculate cust_value (node 528)	calculate_ marketing_vs_ collections (node 532)
"add contribs. of true rules and round up, to max of 10"	"add contribs. of true rules"	"true rule wins; default is 0"	"add contribs. of true rules and round up, to max of 100"	"any true rule gives collect; default is marketing"
frustration_ score	net_profit_ score	late_ payment_ score	cust_ value	marketing_ vs_ collections
NS	SV ⊥	SV 9	NS	NS
READY	DISABLED	VALUE	ENABLED + READY	ENABLED + READY
READY	⊥	⊥	⊥	"collect" C-C
READY	READY	C-V 9	⊥	
	C-V -9	⊥	C-V 10	
	⊥		⊥	
	C-V 50		READY	

09/251998

FIG. 40A

Initialization

Based on the DL specification, compute rows 1, 2, and 3 of the display; 4002

For source attribute cells or row 4 do:
For each source attribute with value, insert value and apply "attribute_value_indication";
For each source attribute that is disabled, apply "attribute_disabled_indication"; 4004

For each non-decision module
In row 5, apply "module_uninitialized_indication";
In row 4, apply "attribute_uninitialized_indication"; 4006

For each decision module
In row 5, apply "module_ready_indication";
In row 4, apply "attribute_uninitialized_indication"; 4008

For each cell in rows 6,7,8,..,apply "rule_ready_indication" 4010

Iteration

For each event of execution engine do
Case on event_type

non_dec_module_enabled:
in row 5, apply "module_enabled_indication"; 4012

non_dec_module_ready:
in row 5, apply "module_ready_indication"; 4014

non_dec_module_ready+enabled:
in row 5, apply "module_ready+enabled_indication"; 4016

non_dec_module_computed:
in row 5, apply "module_computed_indication";
in row 4, label corresponding attribute cell with the value computed and apply
"attribute_computed_indication"; 4018

non_dec_module_value:
in row 5, label cell for this module as "value" and apply "module_value_indication";
in row 4, label corresponding attribute cell with value assigned and apply
"attribute_value_indication" 4020

09/251998

FIG. 40B

non_dec_module_disabled:
in row 5, label cell for this module as "disabled" and apply "module_disabled_indication";
in row 4, label corresponding attribute cell with "⊥" and apply "attribute_disabled_indication" — 4022

dec_module_enabled+ready:
in row 5, label cell with "enabled+ready" and apply "module_enabled+ready_indication"; — 4024

dec_module_computed:
in row 5, label cell with "computed" and apply "module_computed_indication";
in row 4, label cell with the computed value and apply "attribute_computed_indication"; — 4026

dec_module_value:
in row 5, label cell with "value" and apply "module_value_indication";
in row 4, label cell with the computed value and apply "attribute_value_indication"; — 4028

dec_module_disabled:
in row 5, label cell with "disabled" and apply "module_disabled_indication";
in row 4, label cell with "⊥" and apply "attribute_disabled_indication"; — 4030

comp_rule_condition_true:
to corresponding cell, apply "rule_cond_true_indication"; — 4032

comp_rule_contribution_computed:
to corresponding cell, label with computed value and apply "rule_contribution_computed_indication"; — 4034

comp_rule_contributed_value:
to corresponding cell, label with computed value and apply "rule_contributed_value_indication"; — 4036

comp_rule_condition_false:
to corresponding cell, label with "⊥" and apply "rule_condition_false_indication"; — 4038

EndCase